图文 019、 “Stop the World”问题分析：JVM最让人无奈的痛点！

详情 评论

“Stop the World”问题分析

JVM最让人无奈的痛点！

给大家推荐一套质量极高的Java面试训练营课程：

作者是中华石杉，石杉老哥是我之前所在团队的 Leader，骨灰级的技术神牛！

大家可以点击下方链接，了解更多详情，并进行试听：

21天互联网Java进阶面试训练营（分布式篇）

重要说明：

最近不少同学留言反馈，说希望建立一个微信群，供大家进行JVM专栏的学习交流。

这个提议非常好，不过管理微信群是一件挺费时的事儿，我平时工作较忙，实在抽不出时间来进行群管理.

正好石杉老哥的面试训练营建了微信交流群，并且还请了不少一线大厂的助教。

因此跟石杉老哥商量了一下，决定厚着脸皮“鸠占鹊巢”。购买了我JVM专栏的小伙伴，可以加入石杉老哥的微信群，在群里讨论交流技术。

如何加群，请参见文末（注：如果之前已经加过的，就不要重复加群了）

推荐结束，正文开始

1、前文回顾

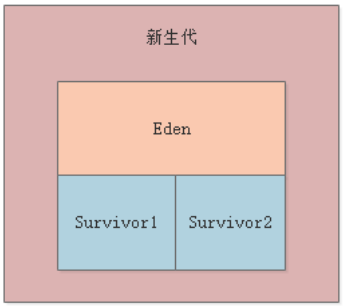
上一篇文章已经通过一个真实的案例分析了新生代的对象分配以及如何转移到老年代，如何频繁触发Full GC的一个场景，同时给出了优化的说明

相信大家通过上一篇文章就已经非常深刻的理解了JVM的核心运行原理了。

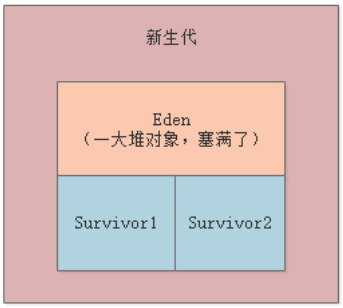
这篇文章我们就来讨论一下基于JVM运行的Java系统，最让我们Java工程师内心痛苦的到底是个什么问题？

2、先来回顾一个新生代GC的场景

大家先来看下面的图，新生代的内存大家都知道是分为Eden和两个Survivor的。

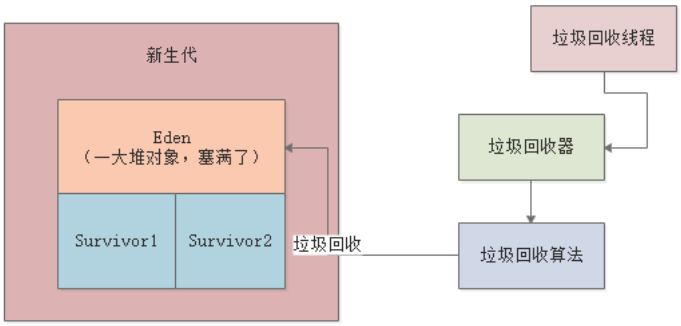


那么此时如果系统不停的运行，然后把Eden给塞满了呢？如下图所示。



这个时候势必就会触发Minor GC了，好，那么之前给大家说过，进行垃圾回收是有专门的垃圾回收线程的，而且对不同的内存区域会有不同的垃圾回收器，大家还记得这个事儿吗？

相当于垃圾回收线程和垃圾回收器配合起来，使用自己的垃圾回收算法，对指定的内存区域进行垃圾回收，大家看看下图。

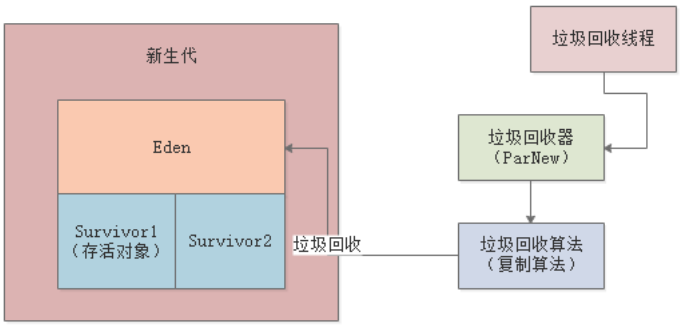


通过上面这个图，大家对垃圾回收线程、垃圾回收器以及垃圾回收算法，是不是就有了一个非常清晰的关系的认识了？

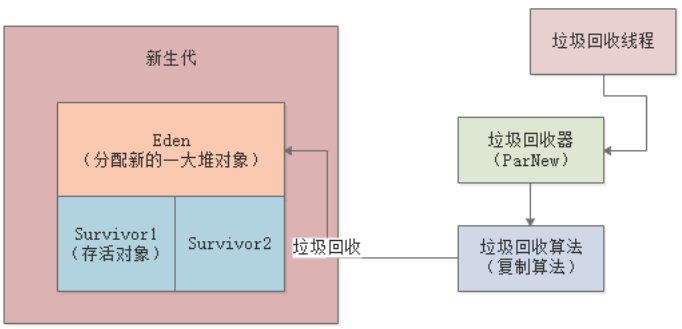
没错，垃圾回收一定会通过一个后台运行的垃圾回收线程来执行他具体的一个逻辑

比如针对新生代我们会用ParNew垃圾回收器来进行回收，然后ParNew垃圾回收器针对新生代采用的就是复制算法来垃圾回收。

这个时候垃圾回收器，就会把Eden区中的存活对象都标记出来，然后全部转移到Survivor1去，接着一次性清空掉Eden中的垃圾对象，如下图。

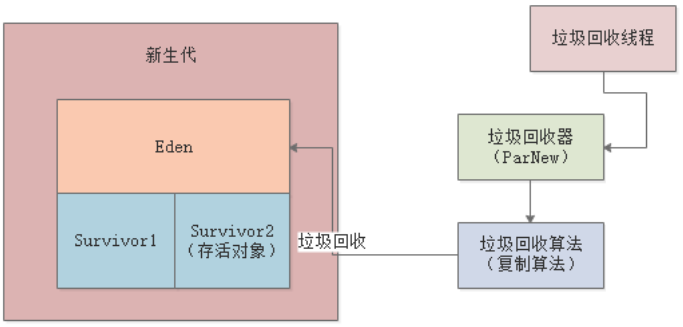


接着系统继续运行，新的对象继续分配在Eden中，如下图所示。



当Eden再次塞满的时候，就又要触发Minor GC了，此时已然是垃圾回收线程运行垃圾回收器中的算法逻辑，也就是采用复制算法逻辑，去标记出来Eden和Survivor1中的存活对象

然后一次性把存活对象转移到Survivor2中去，接着把Eden和Survivor1中的垃圾对象都回收掉，如下图。



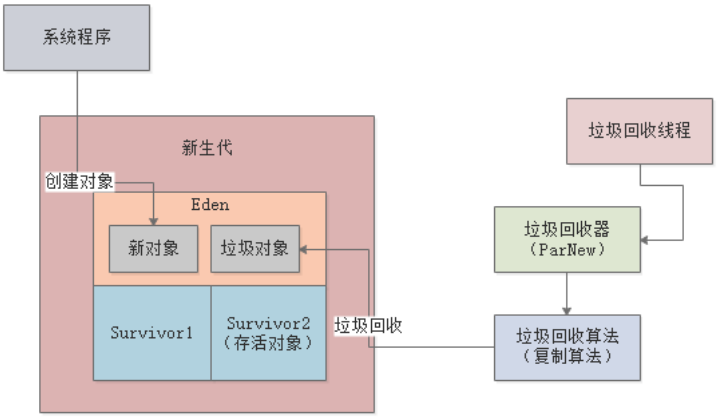
3、GC的时候还能继续创建新的对象吗?

不知道大家有没有考虑过一个问题，之前我们一直都是说GC的原理和JVM整体运行的机制

但是从来没说过在GC的时候，到底我们写好的Java系统在运行期间还能不能继续在新生代里创建新的对象了?

大家可以自己思考一下，假设允许在GC期间，然后还可以继续让系统在新生代的Eden区里创建新的对象，会是一个什么样的场景?

大家看下图。



根据上图所示，如果一边垃圾回收器在想办法把Eden和Survivor2里的存活对象标记出来转移到Survivor1去，然后还在想办法把Eden和Survivor2里的垃圾对象都清理掉，结果这个时候系统程序还在不停的在Eden里创建新的对象。

这些新的对象有的很快就成了垃圾对象，有的还有人引用是存活对象，那现在咋办？

全部乱套了，对于程序新创建的这些对象，你怎么让垃圾回收器去持续追踪这些新对象的状态？

怎么想办法在这次垃圾回收的过程中把新对象中的那些存活对象转移到Survivor2中去？

怎么想办法把新创建的对象中的垃圾都给回收了？

有的同学可能会想当然的说，那就想办法让垃圾回收器来做到啊！

我只能说，大家可以去搞清楚JVM的运行原理，但是不要随意去质疑人家JVM的垃圾回收机制为什么不去那么设计。

因为有些事情想着很简单，但是一旦你要在JVM中去实现的时候，会发现务必的复杂，成本极高，而且很难做到。

所以说，在垃圾回收的过程中，同时还允许我们写的Java系统继续不停的运行在Eden里持续创建新的对象，目前来看是非常不合适的一个事情。

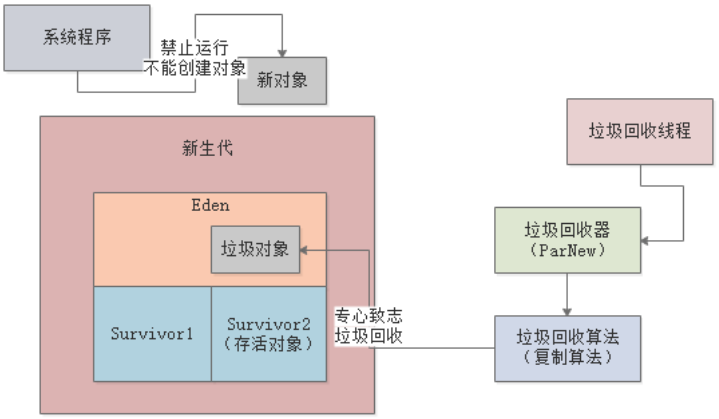
4、JVM的痛点：Stop the World

所以现在大家就好理解了，我们平时使用JVM最大的痛点，其实就是在垃圾回收的这个过程

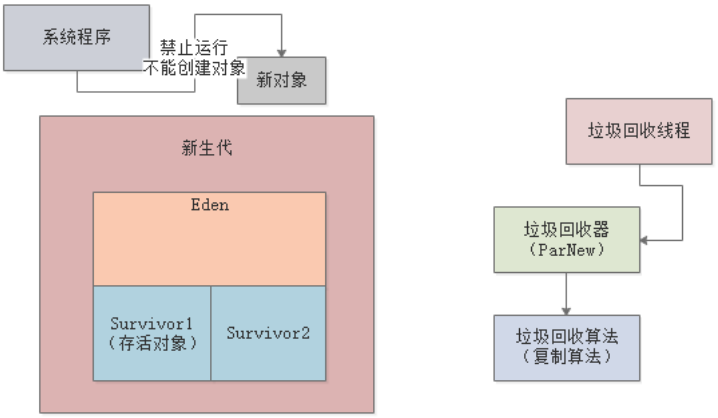
因为在垃圾回收的时候，尽可能要让垃圾回收器专心致志的干工作，不能随便让我们写的Java系统继续对象了，所以此时JVM会在后台直接进入“Stop the World”状态。

也就是说，他会直接停止我们写的Java系统的所有工作线程，让我们写的代码不再运行！

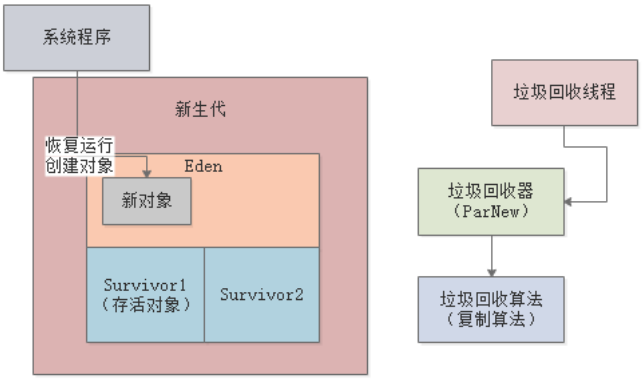
然后让垃圾回收线程可以专心致志的进行垃圾回收的工作，如下图所示。



这样的话，就可以让我们的系统暂停运行，然后不再创建新的对象，同时让垃圾回收线程尽快完成垃圾回收的工作，就是标记和转移Eden以及Survivor2的存活对象到Survivor1中去，然后尽快一次性回收掉Eden和Survivor2中的垃圾对象，如下图。



接着一旦垃圾回收完毕，就可以继续恢复我们写的Java系统的工作线程的运行了，然后我们的那些代码就可以继续运行，继续在Eden中创建新的对象，如下图。



5、Stop the World造成的系统停顿

现在大家就很清晰“Stop the World”会对系统造成的影响了， 假设我们的Minor GC要运行100ms，那么可能就会导致我们的系统直接停顿100ms不能处理任何请求

在这100ms期间用户发起的所有请求都会出现短暂的卡顿，因为系统的工作线程不在运行，不能处理请求。

假设你开发的是一个Web系统，那么可能导致你的用户从网页或者APP上点击一个按钮，然后平时只要几十ms就可以返回响应了

现在因为你的Web系统的JVM正在执行Minor GC，暂停了所有的工作线程，导致你的请求过来到响应返回，这次需要等待几百毫秒。

那么大家可以思考一下，回忆一下上篇文章讲到的案例，因为内存分配不合理，导致对象频繁进入老年代，平均七八分钟一次Full GC，而Full GC是最慢的，有的时候弄不好一次回收要进行几秒钟，甚至几十秒，有的极端场景几分钟都是有可能的。

那么此时一旦你频繁的Full GC，难道你希望你的系统每隔七八分钟就卡死个30秒吗?

在30秒内任何用户的请求全部卡死无法处理，然后用户看到的都是系统超时之类的提示，这会让用户体验极差

所以说，无论是新生代GC还是老年代GC，都尽量不要让频率过高，也避免持续时间过长，避免影响系统正常运行，这也是使用JVM过程中一个最需要优化的地方，也是最大的一个痛点。

6、不同的垃圾回收器的不同的影响

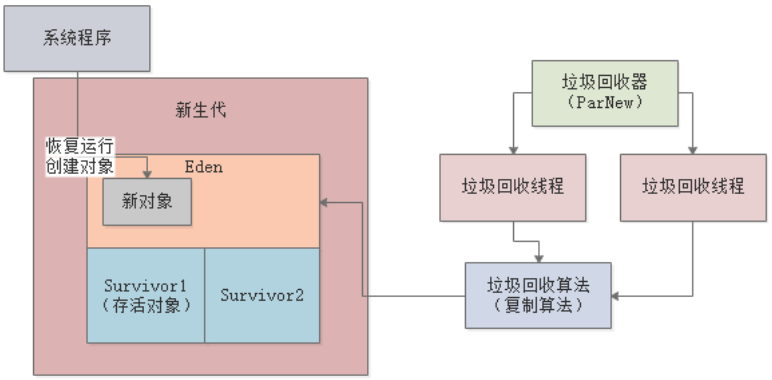
接着今天的话题，再来延伸说一下昨天提到的那些垃圾回收器

比如对新生代的回收，Serial垃圾回收器就是用一个线程进行垃圾回收，然后此时暂停系统工作线程，所以一般我们在服务器程序中很少用这种方式。

但是我们平时常用的新生代垃圾回收器是ParNew，他针对服务器一般都是多核CPU做了优化，他是支持多线程个垃圾回收的，可以大幅度提升回收的性能，缩短回收的时间

所以下周我们深入分析这块的时候，会告诉大家他的很多参数该如何优化

大致原理图如下



大家可以看到，不同的垃圾回收器他会有不同的机制和原理，使用多线程或者单线程，都是有区别的。

然后包括之前给大家提到的CMS垃圾回收器，专门负责老年代的垃圾回收，他也有自己特殊的一套机制和原理，非常的复杂

下周会深入讲CMS垃圾回收器的原理和参数优化，他也是基于多线程的，而且可以使用一套独特的机制尽可能的在垃圾回收的过程中减少“Stop the World”的时间，避免长时间卡死我们的系统。

包括下下周要深入剖析的现在很多公司都在使用的最新的G1垃圾回收器，他更是将采用复杂的回收机制将回收性能优化到机制，尽可能更多的降低“Stop the World”的时间。

其实JVM本身的迭代演进，就是不断的在优化垃圾回收器的机制和算法，尽可能的降低垃圾回收的过程对我们的系统运行的影响。

而我们作为一个合格的Java工程师，我们的责任就是尽可能搞懂这些垃圾回收器的运行机制和算法

然后合理的对线程系统优化内存分配和垃圾回收，尽可能减少垃圾回收的频率，降低垃圾回收的时间，减少垃圾回收对系统运行的影响。

所谓的JVM优化，其实指的就是这个。后续我们会结合大量的案例展开

相信大家一旦坚持3个多月学习完这个专栏，一定会从此脱胎换骨，对JVM的运行原理，垃圾回收机制，然后各种生产故障的监控、排查、定位、分析和解决，都有一个本质的能力提升，在公司里绝对可以搞定自己负责的生产系统的JVM故障。

7、昨日思考题

昨天给大家分析了一个经典的案例，是一个非常经典的真实生产案例和优化实践经验

建议大家不要光看，自己把今天的案例，从背景到分析到解决，一步一步自己画图来推演一遍，彻底吃透这个案例。

这对大家以后分析更多的JVM案例和优化，有非常好的作用。

8、今日思考题

给大家一个小小思考题：到底是单线程进行垃圾回收好呢？还是多线程进行垃圾回收好呢？在不同的场景下有各自的优缺点吗？

大家想想，明天给出答案。

End
